A 3-level Cache Miss Model for a Nonvolatile Extension to Transcendent Memory

Vimalraj Venkatesan Y.C. Tay
National University of Singapore
Singapore

Yi Irvette Zhang
Peking University
China

Qingsong Wei
Data Storage Institute
Singapore

***Abstract*—Resource allocation is fundamental to cloud computing, where the memory hierarchy is deep. Space allocation in this hierarchy calls for a model to determine how provisioning at one level affects performance at a lower level.**

This paper presents a 3-level model that relates the Miss Ratio Curves for two caches at adjacent levels. The model is tested with NEXTmem, which is a transcendent memory used by a Xen hypervisor to cache pages for virtual machines. NEXTmem has a DRAM level and a nonvolatile memory level. The test runs DaCapo benchmarks and shows that the model can be used to enforce fairness at one level, and latency bounds at another level.

***Keywords*-memory hierarchy, cache misses, nonvolatile memory, transcendent memory, virtual machines**

I. Introduction

Resource allocation for virtual machines (VMs) is a fundamental issue in cloud computing. As cores continue to proliferate, the bottleneck for workload consolidation has shifted from cycles to memory [1]. In fact, cloud providers often offer VMs that are specified by how much RAM (random access memory) they are allocated[1]; cores, storage and bandwidth are then scaled proportionately. Moreover, memory has become an issue in power consumption [2].

In response, there are efforts to introduce nonvolatile memory (NVM), to increase capacity and reduce power. There is also a need to distribute storage for big datasets [3]. The result is a memory hierarchy that is many levels deep.

Often, one level in the hierarchy acts as a cache, and thus filter out references, for the next level below. For each level, there is a *Miss Ratio Curve (MRC)* that describes how the miss probability varies with cache size. To understand how the filtering from memory allocation at one level affects performance at a lower level, we need to know: How are MRCs at adjacent levels related?

An MRC is determined by the *interaction* between reference pattern and caching policy. Even for a single level, analyzing this interaction is very difficult, especially since NVM requires replacement policies that are more complicated than LRU (least recently used). Analyzing the interaction between two levels can only be harder.

Fortunately, there is already an equation that can model the MRC for a single level [4]. We use this equation to construct a model for a 3-level hierarchy, and thus analyze the relationship between MRCs for two adjacent levels.

To demonstrate the use of this 3-level model, we consider a para-virtualized system that employs NVM. Specifically, we assume the hypervisor uses *transcendent memory* (tmem [5]), and the NVM resides in tmem.

This paper thus makes two contributions:

(1) We present a 3-level cache miss model that relates the MRCs for two adjacent levels in the memory hierarchy. Sec. II describes this model and applies it to memory provisioning that combines fair allocation and satisfying a Service Level Objective (SLO). The model and the application are validated by simulation using LRU replacement and a reference trace from a real application. This 3-level model can be extended to n levels, for $n > 3$.

(2) We demonstrate in Sec. III how the model can be used to dynamically partition DRAM for one level in tmem and NVM for the next level. (The NVM can use PCM, STT-RAM or some other byte-addressable technology.) This NVM-extended tmem, named **NEXTmem**, uses a tight coupling of replacement policies between the DRAM and NVM levels, and the experiments use real applications running on a NEXTmem implementation in Xen. The set-up is thus a stress test for the model.

Sec. IV surveys related work, before Sec. V concludes with a summary.

II. 3-level Cache Miss Model

This section describes a Cache Miss Equation that can model the unusual replacement policies associated with NVM. Sec. II-A uses it to construct a model for a 3-level memory hierarchy, and Sec. II-B validates it with a real reference trace. Sec. II-C shows how the model can be used to enforce fair allocation and latency SLO, and Sec. II-D tests this application with a simulation.

A. The Cache Miss Equation

Consider three adjacent levels in the memory hierarchy: Level_1, Level_2 and Level_3. Level_1 acts as a cache for Level_2; a reference that misses at Level_1 is forwarded to Level_2. Similarly, Level_2 acts as a cache for Level_3.

[1]E.g. `www.rackspace.com`, `wiki.gogrid.com`

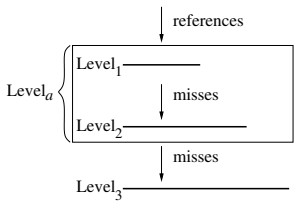


Figure 1. **A 3-level memory hierarchy, with top two levels aggregated as a single level.**

Level_1 and Level_2 may have different cache replacement policies. This can happen if, say, memory is volatile at Level_1, but nonvolatile at Level_2. There are also many possible variations to the cache placement policies. For example, when Level_i gets a page x from Level_j, x may or may not be deleted from Level_j. When a page at Level_3 is copied to Level_1, Level_2 may or may not get a copy.

There is a Miss Ratio Curve MRC_1 for Level_1 and an MRC_2 for Level_2. If we view Level_1 and Level_2 as an aggregated cache called Level_a, as illustrated in Fig. 1, then Level_a gets the same references as Level_1 and suffers the same misses as Level_2. Level_a thus has its own MRC_a. How is MRC_a related to MRC_1 and MRC_2?

Given the many possible combinations of caching policies and infinite variety of real reference patterns, deriving some relationship among the three MRCs seems intractable. Fortunately, this derivation is actually possible if we model each MRC by the following **Cache Miss Equation** [6]

$$\begin{aligned} P^{\text{miss}} &= \frac{1}{2}(H + \sqrt{H^2-4})(P^* + P_c) - P_c \\ \text{where} \quad H &= 1 + \frac{M^* + M_b}{M + M_b} \quad \text{for } M \le M^*. \end{aligned} \tag{1}$$

($P^{\text{miss}} = P^*$ for $M > M^*$.) Here, P^{miss} is the probability of a cache miss for a cache of size M. M^*, M_b, P^* and P_c are parameters with values that depend on the reference pattern and caching policy. These four parameters are minimal, in the following sense:

- P^* is the probability of a *cold miss*, i.e. a miss that brings a page into the cache for the first time. Cold misses are an inherent characteristic of a reference pattern, so any equation for P^{miss} must incorporate P^*.
- When P^{miss} is plotted against cache size M, the points generally trace a decreasing MRC. Previous equations for this MRC, like the power law [7], [8], model this decrease as continuing forever when, in fact, P^{miss} must eventually reach P^* at some $M = M^*$. Any equation for P^{miss} should model this M^*.
- The cache manager needs some memory for its own code and data structures. The space used for such purposes is modeled by M_b in (1).
- P^{miss} is determined by the interaction between the reference pattern and the caching policy. This interaction affects the curve's convexity, and is modeled by P_c.

This Cache Miss Equation (1) was derived using very weak assumptions, so it is generally applicable. It has been extensively validated, at different levels in the memory hierarchy, for widely different systems running a large variety of workloads [4], [6], [9], [10]. We will again validate it in Sec. II-B and Sec. III-D for 2 cache levels.

Since we are considering three MRCs, we use $x \in \{1, 2, a\}$ to index the variables P_x^{miss} and M_x, and parameters M_x^*, M_{xb}, P_x^* and P_{xc} for the MRC at Level_x. In particular, $M_a = M_1 + M_2$.

Note from Fig. 1 that the reference pattern seen by Level_2 are misses from Level_1. The Level_2 MRC therefore depends on Level_1 cache size M_1. In other words, the parameter values at Level_2 may depend on P_1^{miss}.

Let $\langle M_1, M_2 \rangle$ denote Level_a with cache size M_1 at Level_1 and cache size M_2 at Level_2. Level_a has a cache miss if and only if the referenced page is not in both Level_1 and Level_2. Therefore

$$P_a^{\text{miss}} = P_1^{\text{miss}} P_2^{\text{miss}}. \tag{2}$$

Applying the Cache Miss Equation (1) to both P_a^{miss} and P_2^{miss} in (2), we get

$$\begin{aligned} &\frac{1}{2}(H_a + \sqrt{H_a^2-4})(P_a^* + P_{ac}) - P_{ac} \\ &= P_1^{\text{miss}} \left(\frac{1}{2}(H_2 + \sqrt{H_2^2-4})(P_2^* + P_{2c}) - P_{2c} \right) \end{aligned} \tag{3}$$

where

$$H_a = 1 + \frac{M_a^* + M_{ab}}{M_a + M_{ab}} \quad \text{and} \quad H_2 = 1 + \frac{M_2^* + M_{2b}}{M_2 + M_{2b}} \tag{4}$$

When $M_a = M_a^*$ and $M_2 = M_2^*$, we have

$$M_a^* = M_a = M_1 + M_2 = M_1 + M_2^* \tag{5}$$

and $H_a = 2 = H_2$ from (4), so (3) gives

$$\frac{1}{2}(2+0)(P_a^* + P_{ac}) - P_{ac} = P_1^{\text{miss}} \left(\frac{1}{2}(2+0)(P_2^* + P_{2c}) - P_{2c} \right)$$

$$\text{i.e.} \quad P_a^* = P_1^{\text{miss}} P_2^*. \tag{6}$$

A change in M_1 would change the reference pattern seen at Level_2, so M_2^* in (5) varies with M_1, and P_2^* in (6) varies with P_1^{miss}. Substituting (6) into (3), we have

$$\begin{aligned} &\frac{1}{2}(H_a + \sqrt{H_a^2-4})(P_1^{\text{miss}} P_2^* + P_{ac}) - P_{ac} \\ &= \frac{1}{2}(H_2 + \sqrt{H_2^2-4})(P_1^{\text{miss}} P_2^* + P_1^{\text{miss}} P_{2c}) - P_1^{\text{miss}} P_{2c} \end{aligned}$$

One solution to this equation is

$$P_{ac} = P_1^{\text{miss}} P_{2c} \tag{7}$$

$$\text{and} \qquad H_a = H_2. \tag{8}$$

Substituting $M_a = M_1 + M_2$, (5) and (8) into (4), we get

$$1 + \frac{(M_1 + M_2^*) + M_{ab}}{(M_1 + M_2) + M_{ab}} = 1 + \frac{M_2^* + M_{2b}}{M_2 + M_{2b}},$$

$$\text{i.e.} \quad \frac{M_2^* + (M_1 + M_{ab})}{M_2 + (M_1 + M_{ab})} = \frac{M_2^* + M_{2b}}{M_2 + M_{2b}},$$

for which $M_1 + M_{ab} = M_{2b}$ is a solution. We thus have the following solution to (3):

$$\begin{aligned} M_a^* &= M_1 + M_2^* \\ P_a^* &= P_1^{\text{miss}} P_2^* \\ P_{ac} &= P_1^{\text{miss}} P_{2c} \\ M_{ab} &= M_{2b} - M_1 \end{aligned} \tag{9}$$

These equalities relate the three MRCs at Level_a, Level_1 and Level_2. Intuitively, the P_a^{miss} curve can be obtained from the P_2^{miss} curve by the multiplicative factor P_1^{miss} (since $P_a^{\text{miss}} = P_1^{\text{miss}} P_2^{\text{miss}}$) and shifted right by M_1 (since $M_a^* = M_2^* + M_1$).

B. Validation for 3-level model

We now validate the Cache Miss Equation and the relationship (9), using a real page reference trace. This trace is run on a simulated 2-level cache where both levels employ LRU replacement. A page that is evicted from Level_1 is written to Level_2. If a reference misses both Level_1 and Level_2, the page will be retrieved from Level_3 and written to Level_1, without Level_2 getting a copy.

Given a set of $\langle M, P^{\text{miss}} \rangle$ measurements for a workload (i.e. reference pattern plus caching policy), we can use regression to fit the data with the Cache Miss Equation and thus obtain the parameter values. These four values can then be used to plot the curve determined by the equation.

We use `Financial1` from the UMASS Trace Repository[2]. This is a trace of online transactions running at a large financial institution. Fig. 2(a) shows a set of $\langle M_1, P_1^{\text{miss}} \rangle$ measurements for Level_1 from the simulator, together with the curve defined by the Cache Miss Equation, with parameters calibrated by regression against the measurements. The plot shows an excellent fit, with a coefficient of determination $R^2 = 0.9991$ from the regression. (R^2 is a standard statistical measure of how well the equation fits the data; $R^2 = 1$ would indicate a perfect fit.)

For Level_2, we choose $M_1 = 10000$, 20000, 40000, 70000. For each M_1 value, we measure the L_2 misses at various M_2 values. The set of $\langle M_2, P_2^{\text{miss}} \rangle$ measurements for each M_1 value are again fitted with the Cache Miss Equation. Fig. 2(b) shows that the fit is excellent for each M_1. Note that, as M_1 increases, the MRC for Level_2 rises, since a larger M_1 caches more of the hot pages, so Level_2 contains colder pages and suffers more misses.

For cache size $M_a = M_1 + M_2$ at Level_a, one expects the cache contents to depend on M_1. For example, a bigger M_1 may mean more of the cache space is occupied by pages that are cached at both Level_1 and Level_2. Theoretically, one can view Level_a as having a caching policy that depends on M_1 size. It follows that the MRC at Level_a may change when M_1 changes.

However, for the 2-level cache in this simulation, the effect is small. Fig. 2(c) shows that the four sets of $\langle M_1 + M_2, P_a^{\text{miss}} \rangle$ roughly lie on the *same* curve. This means that we can consider the Level_a parameters on the left-hand side of (9) as constants that do not vary with M_1.

C. Applying the Model

We now demonstrate how our 2-level model can be applied to sizing the Level_1 and Level_2 caches for sharing among VMs. We consider the simultaneous satisfaction of two objectives:

System Perspective

Level_1 should be fairly partitioned among the VMs.

User Perspective

Level_a should be sized to satisfy an SLO.

1) System perspective — fair partitioning:

Suppose the Level_1 cache size is M_1 and shared by $\text{VM}_1, \ldots, \text{VM}_k$, $k \geq 2$. Let the Cache Miss Equation's parameters for their Level_1 MRCs be M_{1r}^*, P_{1r}^*, M_{1br} and P_{1cr} for $r = 1, \ldots, k$.

If the Level_1 cache is not overcommitted, then VM_r should be allocated M_{1r}^*; this is sufficient for VM_r to suffer only cold misses, and there is no question of fairness.

Fairness is an issue only when Level_1 is overcommitted and there must be non-cold misses for some or all VMs. How should fairness be defined?

One obvious definition is $P_{11}^{\text{miss}} = \cdots = P_{1k}^{\text{miss}}$, where P_{1r}^{miss} is the Level_1 miss probability for VM_r. However, workloads naturally have different minimum miss probabilities P_{1r}^*, so enforcing equality of P_{1r}^{miss} would unfairly raise the miss probability for a VM that has few cold misses.

Therefore, to make the miss probability for different workloads comparable, we consider the **normalized miss probability** $P_{1r}^{\text{miss}} / P_{1r}^*$; this metric is similar to Kim et al.'s cache fairness metric [11]. We then adopt the following

$$\textbf{fairness definition}: \qquad \frac{P_{11}^{\text{miss}}}{P_{11}^*} = \cdots = \frac{P_{1k}^{\text{miss}}}{P_{1k}^*}. \tag{10}$$

It follows from a result by Tran et al. [6] that a fair allocation of Level_1 cache satisfies

$$M_{1i} + M_{1bi} - \beta_i \sum_{r=1}^{k} (M_{1r} + M_{1br}) = 0$$

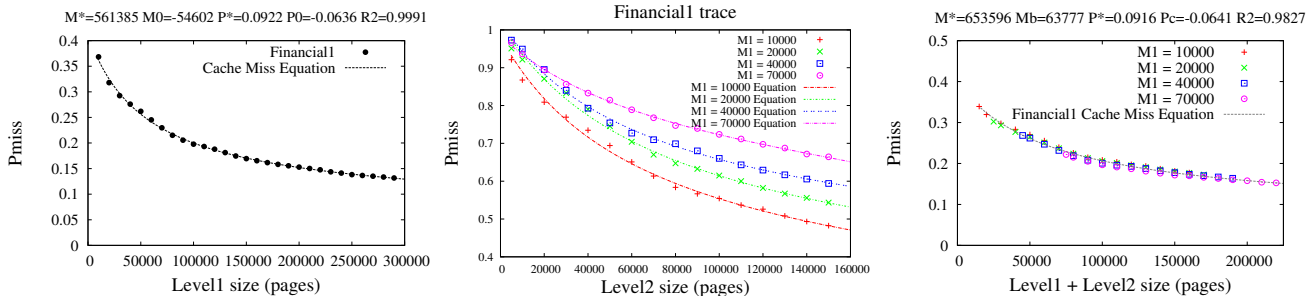
where

$$\beta_i = \frac{\left(\frac{P_{1ci}}{P_{1i}^*} + 1\right)(M_{1i}^* + M_{1bi})}{\sum_{r=1}^{k} \left(\frac{P_{1cr}}{P_{1r}^*} + 1\right)(M_{1r}^* + M_{1br})}$$

Level_1 partitioning implies $\sum_{r=1}^{k} M_{1r} = M_1$, so

$$M_{1i} = \beta_i (M_1 + \sum_{r=1}^{k} M_{1br}) - M_{1bi}. \tag{11}$$

[2] `http://traces.cs.umass.edu/index.php/Storage/Storage`



(a) Level_1 (b) Level_2 for different M_1 values. (c) Level_a: Level_1 plus Level_2.

Figure 2. **Simulation with `Financial1` trace and LRU: The Cache Miss Equation is an excellent fit for the MRCs of Level_1, Level_2 and the aggregated Level_a. The Level_a miss probabilities from four different M_1 values roughly lie on the same curve defined by the equation.**

We can thus *calculate* the partition $M_{11}, \ldots, M_{1k}$ from the parameters M^*_{1r}, P^*_{1r}, M_{1br} and P_{1cr}, without need for control loops and gradient descent [12], [13].

2) User perspective — service level objective: SLO is often formulated as a percentile for latency [14]. Let T be the latency for a call on Level_a to retrieve a page x. We state the SLO as

$$\text{Prob}(T > L_{\max}) < p_{\max}, \tag{12}$$

i.e. the probability is at most $p_{\max}$ that T exceeds a given bound $L_{\max}$. Let $y \in \text{Level}_x$ denote y is in Level_x. We assume that every page y can always be found in Level_1 or Level_2 or Level_3. Let L_x be the latency for retrieving a page from Level_x. Then

$$\begin{aligned}&\text{Prob}(T > L_{\max})\\ &= \text{Prob}(y \in \text{Level}_1)\text{Prob}(L_1 > L_{\max})\\ &+ \text{Prob}(x \notin \text{Level}_1 \text{ and } x \in \text{Level}_2)\text{Prob}(L_2 > L_{\max})\\ &+ \text{Prob}(x \notin \text{Level}_1 \text{ and } x \notin \text{Level}_2)\text{Prob}(L_3 > L_{\max}).\end{aligned} \tag{13}$$

Here, we consider the case where Level_3 is storage (e.g. disks), with L_3 exceeding L_1 and L_2 by orders of magnitude. An SLO would specify $L_{\max}$ greater than the average L_3; since $L_3 >> L_1$ and $L_3 >> L_2$ (on average), we have

$$\text{Prob}(L_1 > L_{\max}) \approx 0 \approx \text{Prob}(L_2 > L_{\max}).$$

We thus get from (12) and (13),

$$p_{\max} > P^{\text{miss}}_a \text{Prob}(L_3 > L_{\max}). \tag{14}$$

As $P^{\text{miss}}_a \geq P^*_a$, (14) thus requires $p_{\max}$ and $L_{\max}$ to satisfy $P^*_a \leq p_{\max}/\text{Prob}(L_3 > L_{\max})$. To satisfy (14), we set

$$P^{\text{miss}}_a = \frac{0.95 p_{\max}}{\text{Prob}(L_3 > L_{\max})}. \tag{15}$$

Given $p_{\max}$, $L_{\max}$ and the cumulative distribution function (CDF) for L_3, we can use (15) to calculate P^{miss}_a and then solve Level_a's Cache Miss Equation to determine M_a.

Suppose there are multiple VMs, as in Sec. II-C1, and the SLO (12) is specified for VM_r. Then, (15) determines M_{ar}. If Level_1 allocation is determined to be M_{1r} by fair allocation, then the Level_2 allocation is

$$M_{2r} = M_{ar} - M_{1r}, \tag{16}$$

thus combining the user perspective (M_{ar}) and the system perspective (M_{1r}).

D. Validation for the SLO application

We will test the use of the Cache Miss Equation for the combination of fair partitioning and latency SLO in Sec. III-D. In this section, we first test how the SLO (12) can be satisfied for dynamically varying workload.

It is common for a server to host homogeneous VMs, with each VM running a single application, and the application is the same for all VMs. Let k be the number of VMs, and $M_a(k)$ the total Level_1 and Level_2 allocation for these k VMs. The Level_a miss probability depends on k, so we have a Miss Ratio Curve $\text{MRC}_a(k)$ for each k.

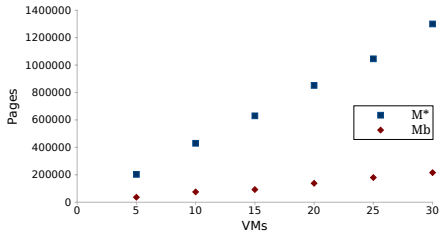
We can fit each $\text{MRC}_a(k)$ with the equation to get corresponding $M^*_a(k)$, $M_{ab}(k)$, $P^*_a(k)$ and $P_{ac}(k)$. How do these parameters depend on k?

To find out, we simulate multiple VMs, each running `Financial1` and with both Level_1 and Level_2 using LRU. Fig. 3 shows that $M^*_a(k)$ and $M_{ab}(k)$ vary linearly with k, whereas $P^*_a(k)$ and $P_{ac}(k)$ are approximately constant.

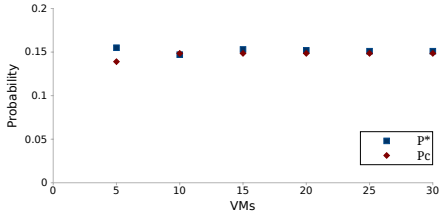
Next, consider a scenario where the hypervisor is hosting a tenant that runs k such VMs, and k changes dynamically. We simulate the VMs with k concurrent threads, starting with $k = 10$. When a VM finishes running the trace, it terminates. Time is divided into 1-minute epochs. At the end of each epoch, we add another 2 VMs, and use the resulting k value to determine $M^*_a(k)$, $M_{ab}(k)$, $P^*_a(k)$ and $P_{ac}(k)$ from Fig. 3 by interpolation.

Suppose the SLO requires $M_2(k) = 2M_1(k)$, $L_{\max} =$ 300ms and $p_{\max} = 0.03$. We measure the CDF of the disk latency for `Financial1` and find $\text{Prob}(L_3 > L_{\max}) =$ 0.135. By (15), we require $P^{\text{miss}}_a(k) = 0.211$.

At the start of each epoch, we substitute $P^{\text{miss}}_a(k) = 0.211$ into the Cache Miss Equation to determine $M_a(k)$ and thus $M_1(k) = M_a(k)/3$ and $M_2(k) = 2M_a(k)/3$. For the rest of

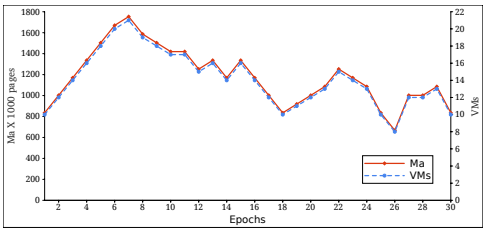


(a) $M_a^*(k)$ and $M_{ab}(k)$ increase linearly with k.

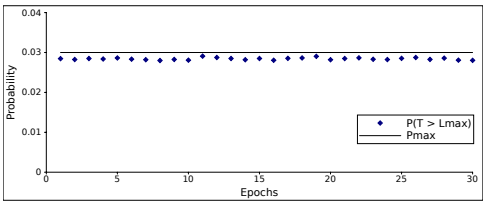


(b) $P_a^*(k)$ and $P_{ac}(k)$ are constant with respect to k.

Figure 3. **How the** Level_a **parameters vary with the number of VMs** k**, for homogeneous VMs running** `Financial1`**.**



(a) How k and $M_a(k)$ change dynamically.



(b) The SLO bound $p_{\max} = 0.03$ is always satisfied.

Figure 4. **In (a), the right vertical axis is for number of VMs** k **and the left vertical axis is for** $M_a(k)$**; (b) shows that the fraction of** `get` **latency exceeding** $L_{\max}$ **is always below** $p_{\max}$**.**

the epoch, we use the CDF to generate disk latency for each Level_a miss, and count how many `gets` have $T > L_{\max}$.

Fig. 4(a) plots how k varies over time and how the calculated allocation $M_a(k)$ changes in tandem. Fig. 4(b) shows that, despite the changes in k, the equation-based dynamic allocation of $M_1(k)$ and $M_2(k)$ does keep the latency T below the SLO bound (12).

III. NEXTmem: An NVM extension to transcendent memory

Sec. III-A first describes tmem before Sec. III-B explains how it is re-implemented to include NVM. Sec. III-C describes our experimental set-up that we use in Sec. III-D to test the application of the 3-level model to NEXTmem.

A. Transcendent memory

Transcendent memory (tmem) is a hypervisor-managed cache for pages that are evicted by guest VMs. The current implementation of tmem has a *cleancache* for caching clean pages that a VM would otherwise discard, and a *frontswap* for caching dirty pages that a VM would otherwise write to disk. This caching reduces disk reads for pages that the VMs need again, and reduces disk writes for swapping.

The hypervisor needs a pool of pageframes for tmem to contain the pages it wants to cache. Some of these pageframes are obtained by ballooning to collect idle memory from VMs [15]. The hypervisor also adds to the pool those pageframes that it has not yet allocated to VMs.

When a VM wants to evict a page x from its memory, it calls tmem to **put** x there. This `put` can fail, as when tmem is full. If it succeeds, the VM gets an identifier for x.

If the VM later has a page fault on x, it uses the identifier to **get** it from tmem. If x is clean, this `get` can fail, as tmem may discard x if it needs to free up some pageframes; cleancache is thus *ephemeral*. If x is dirty, the `get` will always succeed, so frontswap is *persistent*.

The above is a description of tmem as currently implemented [5]. We next describe the re-implemented version.

B. NEXTmem

RAM now plays a central role in resource provisioning for VMs. DRAM prices increase rapidly beyond 64GB, and it is also a major drain on power. There is therefore much research and development now on augmenting DRAM with NVM [**?**], [**?**], since it is cheaper and uses less power.

However, NVM can support far fewer writes than DRAM before they wear out. We therefore control access to NVM by not allocating it to VMs but, instead, let the hypervisor manage it. Specifically, we put NVM in tmem.

An obvious use of NVM is for it to host frontswap. Swap space is used sparingly by a system that is properly provisioned, so there should be plenty of NVM leftover from frontswap. We therefore use the excess NVM for cleancache.

Besides endurance, another issue with NVM is latency, since it is slower than DRAM. We therefore use the NVM cleancache for cold pages, and use DRAM to cache hot pages. We are thus led to a *redesign* of tmem.

Fig. 5 illustrates our design of the architecture for extending tmem to include NVM. We provide enough NVM to accommodate the recommended swap space (e.g. 8GB for a 16GB RAM[3]), so a dirty `put` will always succeed (for a well-behaved system). The part of NVM that caches dirty pages is called *swap region* (equivalently, frontswap).

We split cleancache into two levels, one in DRAM and one in NVM. A successful `put` of a clean page x results in x

[3]http://access.redhat.com/documentation

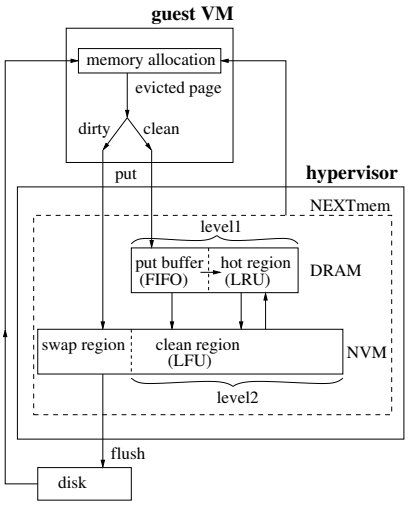


Figure 5. **Architecture of NEXTmem. Arrows indicate writes. A** `get` **results in a read from NEXTmem or from disk.**

joining a FIFO (first-in-first-out) queue at the DRAM level. This queue serves as a *put buffer*. When the DRAM level is full and another `put` arrives, the oldest page is transferred to a *clean region* at the NVM level.

If there are two `get`s for x while it is in the put buffer, x is transferred to a *hot region* at the DRAM level and pushed onto the top of an LRU stack. A `get` on a page y in this stack will move y to the top. The pages at the stack bottom are cold; when the DRAM level is full, the bottom page is transferred to the clean region at the NVM level.

The clean NVM region uses LFU replacement, so the least frequently used page is discarded if the NVM level is full. If there are two `get`s for a page z in this clean region, z will be copied to the hot region at DRAM level.

The justification for this design, and experimental results to demonstrate its efficacy, are provided in a separate paper [16]. For this paper, our focus is on using the 3-level cache model in Sec. II to analyze the behavior of the cleancache hierarchy — a Level_1 that consists of put buffer and hot region in DRAM, a Level_2 that is the clean region in NVM, and a Level_3 that is the disk.

It would seem intractable to model the unusual and tightly coupled replacement policies (a mix of FIFO, LRU and LFU; a transfer or copy within DRAM or between DRAM and NVM). The experimental results that follow show that our 3-level model can in fact overcome this difficulty.

C. Experimental Set-up

The NEXTmem experiments are run on a machine that has an Intel Xeon 2.5GHz processor with 12 cores, 2×4GB of DRAM and a SATA 2.0, 1TB, 5400 rpm Seagate disk. The hypervisor is Xen 4.1.3, with its tmem interface (for `put`, `get`, etc.) unchanged, but the frontswap and cleancache inside are replaced by NEXTmem.

VM_1:	TS; TS; XL; XL; EC; XL; XL; TS; H2; XL; H2; TS; TB; TS; XL;
VM_2:	XL; TB; TS; H2; TS; JY; XL; TS; JY; XL; TS; XL; TB; TB; XL;
VM_3:	H2; TB; XL; XL; XL; TS; XL; TS; TB; TB; TS; EC; TS; TB; JY;
VM_4:	H2; TS; TS; H2; EC; JY; H2; EC; XL; EC; XL; XL; TB; XL; TB;
VM_5:	JY; EC; JY; XL; XL; H2; H2; EC; H2; XL; TB; TS; EC; XL; XL;

Table I
Sequence of applications run by each VM for the experiment in Sec. III-D. (TS=`Tradesoap`, TB=`Tradebeans`, H2=`H2`, XL=`Xalan`, JY=`Jython`, EC=`Eclipse`.)

To apply pressure on NEXTmem, we set $M_1 = 70000$ 4KByte pages (put buffer plus hot region). The NVM has 300000 pages, so M_2 is 300000 minus the swap region. This M_1 and M_2 are shared by the VMs.

There is no commodity byte-addressable NVM that we could use for this experiment, so the NVM is emulated by DRAM. For the SLO considered here, the DRAM and NVM latencies are negligible when compared to disk latency.

Each VM has 2 vCPU, 500MB of DRAM and a 100GB virtual disk, and runs Ubuntu 12.04 LTS.

D. Application of 3-level model to NEXTmem provisioning

The validation of the Cache Miss Equation in Fig. 2 is for a trace simulation over two LRU caches. We now confirm that the equation does work with a real set-up; specifically, we have a VM running the DaCapo `Tradebeans` benchmark[4] on the machine with the NEXTmem implementation.

Fig. 6(a) shows that the equation is an excellent fit for the DRAM Level_1 in NEXTmem, and Fig. 6(b) shows similarly good fit for the NVM Level_2 at different M_1 values. Like in Fig. 2(c), the four sets of $\langle M_1 + M_2, P_a^{\text{miss}} \rangle$ values again lie on the same curve defined by the equation in Fig. 6(c).

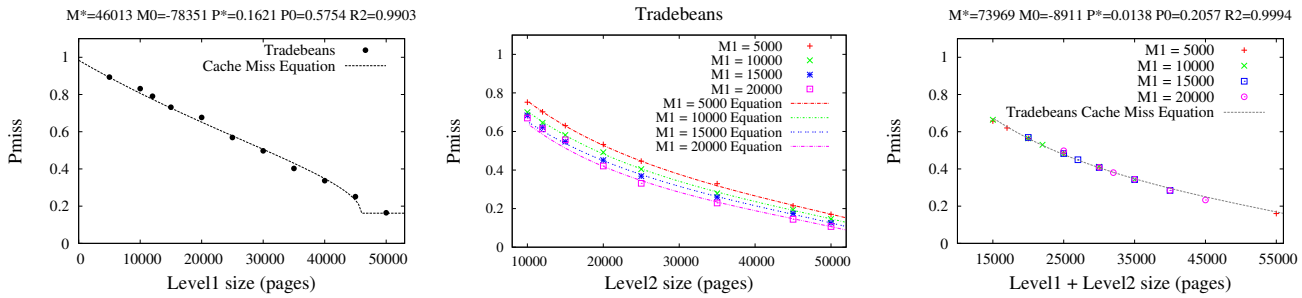
Space constraint prevents us from presenting more results from experiments validating the equation for NEXTmem.

To illustrate the application of the 3-level model, we run 5 concurrent VMs. Each VM runs a sequence of randomly chosen DaCapo benchmarks, so the VM goes through phases when it is more memory-intensive than usual. The VMs run different sequences, as shown in Table I, so the VMs are not identical in their memory demand.

The SLO is $p_{\max} = 0.05$ and $L_{\max} = 300$ms for VM_1, and $p_{\max} = 0.02$ and $L_{\max} = 500$ms for VM_2. There is no SLO for the other 3 VMs.

Time is divided into 90-second epochs. The experiment starts with M_1 and M_2 both equally partitioned among the 5 VMs. After each epoch, there is a 4-second calibration period, during which we vary each VM's M_1 and M_2 allocation and measure their miss probability; the results are then used for regression to obtain the parameter values for the Cache Miss Equation for each level and each VM. This periodic recalibration is necessary because the VMs' need for memory is changing dynamically.

[4] `http://www.dacapobench.org`



(a) Level_1 (b) Level_2 for different M_1 values. (c) Level_a: Level_1 plus Level_2.

Figure 6. **Experiment with DaCapo `Tradebeans` and NEXTmem: The Cache Miss Equation is an excellent fit for the MRCs of Level_1, Level_2 and the aggregated Level_a. The Level_a miss probabilities from four different M_1 values roughly lie on the same curve defined by the equation.**

After recalibration, we use the parameters and (11) to fairly partition M_1 (Sec. II-C1). We then calculate the Level_a allocation for VM_1 and VM_2 to satisfy their SLO (Sec. II-C2) and, by (16), their Level_2 allocation. The remaining Level_2 space is then equally partitioned among VM_3, VM_4 and VM_5.

We measure P^{miss}/P^* for each VM in each epoch, and the disk latency for each Level_a miss.

Fig. 7(a) plots the resulting M_1 and M_2 allocation to the 5 VMs as time passes. Fig. 7(b) shows that the P^{miss}/P^* values are similar for Level_1, matching the fairness definition (10). According to Jain's fairness index, the fairness in Fig. 7(b) varies from 0.96 to 0.99 (1 indicates perfect fairness). Furthermore, Fig. 7(c) shows that the Level_a allocation satisfies the latency SLO for VM_1 and VM_2.

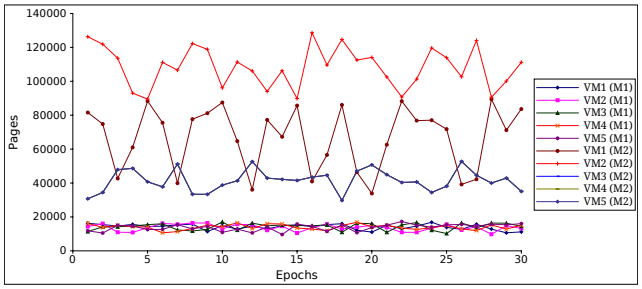
IV. Related Work

The power law $P^{\text{miss}} = aM^b$ is the equation that is often used to model MRCs [7]. Other alternatives include exponential and logarithmic functions [17]. Such functions for P^{miss} are strictly monotonic, so they do not identity an M^* at which P^{miss} measures only cold misses.

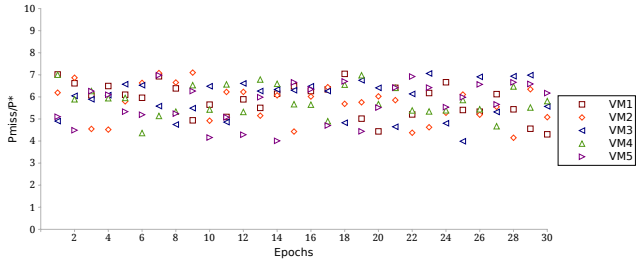
Modeling an MRC mathematically is very difficult, since it is determined by the interaction between reference pattern and cache policy. If we consider two levels in the memory hierarchy, then the reference pattern seen by the lower level is determined by the cache size *and* the replacement policy at the upper level. This filtering effect compounds the difficulty [3], even under strong assumptions [8].

We overcome this difficulty by using the Cache Miss Equation (1). This equation is based on weak assumptions [4], so it works in widely different contexts [6], [9], [10]. They key idea is to parameterize the equation, and have the parameter values determined by the context.

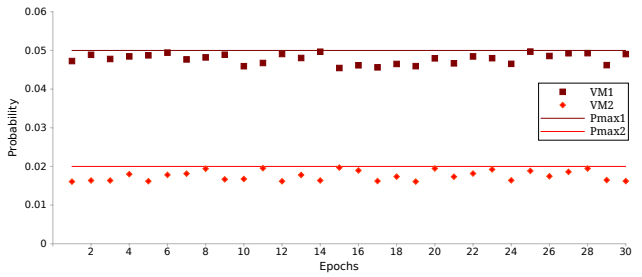
Having a Cache Miss Equation that can adapt to different contexts is especially necessary because there are many ways NVM can be introduced into the memory hierarchy. It can come between DRAM and disk, so the DRAM layer acts as a cache for the NVM layer [18], or the two can be parallel [19]. NVM allocation can be determined by



(a) Dynamic allocation of M_1 and M_2 to 5 VMs.



(b) Fair Level_1 P^{miss}/P^* values for the 5 VMs.



(c) Level_a allocation enforces SLO for VM_1 and VM_2.

Figure 7. **Dynamic DRAM and NVM partitioning for NEXTmem. VM_3, VM_4 and VM_5 have the same M_2 allocation in (a).**

hardware [20], managed by the operating system [21], or supported by a programming language [22].

In every of these designs, NVM calls for a management policy that takes endurance into account. This generates

many new replacement policies, but the Cache Miss Equation can model each through parameter calibration.

This paper considers Venkatesan et al.'s Ex-Tmem design for introducing NVM into hypervisor memory via tmem [16]. The introduction of NVM requires a revamp of Magenheimer's tmem design [5]. In particular, Ex-Tmem — called NEXTmem in this paper — has a two-level cache with an elaborate management policy. NEXTmem is thus a stress test for both the Cache Miss Equation and the 3-level model presented here.

V. Conclusion

We have presented a 3-level model for the relationship among caches in a hierarchy. If there is a Level_4, then this model can be applied to Level_a, Level_3 and Level_4. In this way, the model can be generalized to n levels, for any $n > 3$. The memory hierarchy in a cloud system is deep, so it needs such a n-level model that can handle the filtering effect and heterogeneous caching policies.

We have tested the model with an NVM-extended transcendent memory for virtual machines. The results show that we can use it effectively to dynamically partition two levels, combining fair allocation and SLO satisfaction.

Acknowledgment

We would like to thank the anonymous reviewers for their feedback and suggestions for improvement. This work was supported by Agency for Science, Technology and Research (A*STAR), Singapore under Grant No. 112-172-0010.

References

[1] D. Gupta, S. Lee, M. Vrable *et al.*, "Difference engine: harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, 2010.

[2] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, 2nd Edn.* Morgan & Claypool, 2013.

[3] X. Li, A. Aboulnaga, K. Salem, A. Sachedina, and S. Gao, "Second-tier cache management using write hints," in *Proc. FAST*, December 2005, pp. 115–128.

[4] Y. C. Tay and M. Zou, "A page fault equation for modeling the effect of memory size," *Perform. Eval.*, vol. 63, no. 2, pp. 99–130, 2006.

[5] D. Magenheimer, "Transcendent Memory on Xen," Xen Summit (`http://vimeo.com/13005922`), 2009.

[6] D. N. Tran, P. C. Huynh, Y. C. Tay, and A. K. H. Tung, "A new approach to dynamic self-tuning of database buffers," *ACM Trans. Storage*, vol. 4, no. 1, pp. 1–25, 2008.

[7] L. A. Belady, "A study of replacement algorithms for virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[8] A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma, "Cache miss behavior: is it $\sqrt{2}$?" in *Conf. Computing Frontiers*, 2006, pp. 313–320.

[9] M. Rezazad and Y. C. Tay, "A cache miss equation for partitioning an NDN content store," in *Proc. Asian Internet Engineering Conference*, 2013, pp. 1–8.

[10] Y. C. Tay, X. Zong, and X. He, "An equation-based heap sizing rule," *Perform. Eval.*, vol. 70, pp. 948–964, 2013.

[11] S. Kim, D. Chandra, and Y. Solihin, "Fair cache sharing and partitioning in a chip multiprocessor architecture," in *Proc. PACT*, Washington, DC, USA, 2004, pp. 111–122.

[12] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao, "An adaptive control framework for QoS guarantees and its application to differentiated caching services," in *Proc. IWQoS*, Miami Beach, FL, 2002, pp. 23–32.

[13] G. Soundararajan, D. Lupei, S. Ghanbari *et al.*, "Dynamic resource allocation for database servers running on virtual storage," in *Proc. FAST*, 2009, pp. 71–84.

[14] C. Stewart, A. Chakrabarti, and R. Griffith, "Zoolander: Efficiently meeting very strict, low-latency SLOs," in *Proc. ICAC*, June 2013, pp. 265–277.

[15] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. OSDI*, 2002, pp. 181–194.

[16] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," in *Proc. HotOS*. USENIX Association, 2009, pp. 14–14.

[17] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," *ASPLOS*, pp. 91–104, 2011.

[18] V. Venkatesan, Q. Wei, and Y. C. Tay, "Ex-Tmem: Extending transcendent memory with non-volatile memory for virtual machines," in *Proc. HPCC DATA Workshop*, August 2014.

[19] G. Chockler, G. Laden, and Y. Vigfusson, "Data caching as a cloud service," in *Proc. Int. Workshop on Large Scale Distributed Systems and Middleware*, 2010, pp. 18–21.

[20] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proc. ISCA*, 2009, pp. 24–33.

[21] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. DAC*, 2009, pp. 664–469.

[22] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, "Row buffer locality aware caching policies for hybrid memories," in *Proc. ICCD*, Oct. 2012, pp. 337–344.

[23] L. Wang, H. Wang, L. Cai, R. Chu, P. Zhang, and L. Liu, "A hierarchical memory service mechanism in server consolidation environment," in *Proc. ICPADS*, 2011, pp. 40–47.

[24] G. Nakagawa and S. Oikawa, "Language runtime support for NVM/DRAM hybrid main memory," in *Proc. IEEE COOL Chips XVII*, April 2014.